DOCUMENT RESUME

ED 329 220 IR 014 859

AUTHOR McArthur, David
TITLE Developing Computer Tools To Support Performing and Learning Complex Cognitive Skills. A RAND Note.
INSTITUTION Rand Corp., Santa Monica, Calif.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO RAND-N-2980-NSF'
PUB DATE Jul 89
NOTE 22p.
PUB TYPE Viewpoints (Opinion/Position Papers, Essays, etc.) (120) -- Reports - Descriptive (141)

EDRS PRICE MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS Algebra; *Computer Assisted Instruction; *Computer Software; High Schools; Learning Processes; Microcomputers; *Problem Solving; Student Motivation; Teaching Methods; *Thinking Skills

ABSTRACT

The main aim of this paper is to demonstrate that new and highly effective computer-based learning tools can be designed by adhering to a simple principle: Good learning tools conform to and support the processes and structures that comprise learning. The paper first discusses the processes involved in learning cognitive skills, then describes several software tools that support and facilitate these skills. The examples discussed are drawn from learning problem-solving skills in high school algebra, and learning how to play the strategic board game of Go. Although some of the tools described embed considerable complex intelligence, many are relatively simple to implement and are easily within the current state of the art of computer hardware and software. (16 references) (Author)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

IR

ED329220

A RAND NOTE

Developing Computer Tools to Support Performing and Learning Complex Cognitive Skills

David McArthur

July 1989

BEST COPY AVAILABLE

"PERMISSION TO REPRODUCE THIS MATERIAL IN MICROFICHE ONLY HAS BEEN GRANTED BY

Phyllis Switzer

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

TR 014 859

RAND

The research described in this report was sponsored by the National Science Foundation under Grant No. MDR-8751515.

This Note contains an offprint of RAND research originally published in a journal or book. The text is reproduced here, with permission of the original publisher.

The RAND Publication Series: The Report is the principal publication documenting and transmitting RAND's major research findings and final research results. The RAND Note reports other outputs of sponsored research for general distribution. Publications of The RAND Corporation do not necessarily reflect the opinions or policies of the sponsors of RAND research.

Published by The RAND Corporation
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138

A RAND NOTE

N-2980-NSF

Developing Computer Tools to Support Performing and Learning Complex Cognitive Skills

David McArthur

July 1989

Prepared for
The National Science Foundation

RAND

13 Developing Computer Tools to Support Performing and Learning Complex Cognitive Skills

David McArthur
Rand Corporation

ABSTRACT

The main aim of this paper is to demonstrate that new and highly effective computer-based learning tools can be designed by adhering to a simple principle: Good learning tools conform to and support the processes and structures that comprise learning. I first discuss the processes involved in learning cognitive skills, then describe several software tools that support and facilitate these skills. The examples I discuss are drawn from learning problem-solving skills in high school algebra, and learning how to play the strategic board game of Go. Although some of the tools described embed considerable complex intelligence, many are relatively simple to implement and are easily within the current state of the art of computer hardware and software.

INTRODUCTION

The computer is a demanding tool. Unlike most pieces of technology it has no single purpose, and because it can be used in so many ways, it can be badly misused. Educational applications of computers are a good case in point. Most educational software does a poor job of helping us learn, and many of the programs that do have educational value do not effectively exploit the computer medium. They embed teaching techniques developed in and for a pencil-and-paper educational environment. The mindless translation of educational materials from traditional media to the computer is especially unfortunate because computers have the potential to be much better learning tools than pencil and

This research is being funded by the National Science Foundation (Applications of Advanced Technologies Program). Views or conclusions expressed herein do not necessarily represent the policies or opinions of the sponsor. Reprinted from Applications of Cognitive Psychology: Problem Solving, Education, and Computing, K. Pezek, D. Berger and B. Bankes (Eds.), pp. 183–200, **© 1987 by Lawrence Erlbaum Associates, Inc. Reprinted by permission.**

paper. To begin to develop outstanding educational software we must stop using past techniques as a guide, and having thrown away the old guides, we must find new principles to follow.

The main aim of this paper is to indicate some new guiding principles, and to show that, using them, it is now possible to design very useful and novel learning tools that exploit unique properties of the computer medium. The paper discusses examples of how computers can provide *interactive practice tools* for the development of complex cognitive skills. The examples illustrate that many practice environments are easily within the current state of the art of hardware and software. The tools described obtain their leverage not only through complex, intelligent software, but also through simple software intelligently conforming to, and guided by, the *processes* and *structure* behind *cognitive skills*.

LEARNING THROUGH PRACTICE

Cognitive skills, like how to speak, play chess, or solve algebra problems, can only be learned well by practicing them. To improve our speech we make utterances, to improve our chess we play games, to improve our algebra we solve problems. Clearly, to aid in the learning of cognitive skills we need effective practice environments. Unfortunately, most traditional and computer-based environments do not promote rapid learning through practice. The main reason we possess few good learning aids is that we have few principles to guide us in the development of educational aids. The wisdom of hindsight, more than foresight, tells us why some educational experiments failed (e.g., the "new math"). We need principles that will act as criteria for the design of educational tools and that will allow us to understand *in advance* why one kind of aid would be more effective than another.

One principle I suggest is that the design of learning aids should be based on an understanding of how people perform cognitive skills and how they learn from practice. This principle should not be controversial. It is accepted that tools aiding the performance of a mechanical skill (e.g., building a table) facilitate the specific component activities or processes of the skill (e.g., T-squares make it easier to get right-angled corners). Once we accept the idea that cognitive skills, analogous to mechanical skills, comprise specific information processing steps, then referring to educational aids as tools ceases to be an idle metaphor. Educational materials should be viewed as tools that aid the performance of cognitive skills precisely because, to be successful, they should facilitate the component activities that make up the skill.

Learning through practice is itself a cognitive skill. Consequently, learning tools should not just be designed to help practice as an undifferentiated whole; they should facilitate the particular information processing activities that we know contribute to learning cognitive skills. Brown (1984) has also noted the importance of tailoring software aids to the cognitive processes of the user, and

Shavelson (1981), among others, has pointed out several contributions of cognitive research to the teaching and learning of mathematics.

Until recently we have not been able to put this principle into operation because the processes behind the practice and learning of cognitive skills have been a mystery. In the past decade, however, work in cognitive psychology and artificial intelligence on *learning-by-doing* has produced parts of a formal theory of how students might learn through practice (Anderson, 1982; Brown & Burton, 1978; Brown & Van Lehn, 1980). This theory is still fragmentary, and developing a more complete theory of learning through practice remains a high priority task in designing an environment for learning cognitive skills. But it is not necessary to wait for a fully mature theory before beginning to develop useful tools to support learning. A modest consensus picture of learning through practice is beginning to emerge. I believe that keeping in mind just a few of these consensus, "commonsense," notions of learning through practice will enable us to design some surprisingly powerful educational aids.

These notions should assist in creating and evaluating educational tools developed for any medium; however, they should be especially useful in helping us to develop educational software. Because the computer is potentially a reactive and interactive medium, it is in a unique position to support the processes associated with learning and performing cognitive skills. In the next section I describe the basic processes of learning through practice, mentioning especially where they become difficult and go astray. The following sections discuss a variety of computer-based education tools that students can use to simplify learning processes.

A Commonsense View of Learning Through Practice

Common sense tells us that learning a complex skill through practice is a cyclic process that begins when the student is given a *task* or *problem*, and uses his or her formative *knowledge* of the skill to produce a line of *reasoning* that gives an observable result or *answer*. Knowledge that contributes to an answer falls into one of two broad types. *Factual* knowledge encodes declarative information specific to the problem domain. *Planning knowledge* encodes information about how to use the factual truths of the domain in actually reasoning toward a task goal. Planning knowledge can include both general, domain-independent, problem-solving methods (e.g., heuristic search), and highly domain-specific reasoning techniques. Typically students already know general methods when they are first exposed to a domain, and initially use them to accomplish tasks. However, although these methods are general, they are also often weak and inefficient at producing solutions. This inefficiency motivates students to acquire less generally applicable methods that make problem solving much more rapid.

The student often selects and applies knowledge repeatedly to a task, each iteration yielding one or more reasoning steps. Once the student has exercised his formative skills and has given an answer (or a partial answer; he need not

actually complete his reasoning before deciding he is on the wrong track), he needs to collect information that will allow him to *diagnose* success or failure. Diagnosis goes well beyond simply determining whether the student is wrong or right. If students cannot quickly determine *why* they were wrong (or inefficient) they have little chance to improve their knowledge. Therefore, the process of diagnosis involves *detecting* the specific reasoning step(s) in error, and thus the piece(s) of knowledge that caused an overt mistake, amongst all the knowledge used in reasoning. If the student's answer is correct, diagnosis, if done at all, takes on the character of a review of the formative knowledge that has contributed to the answer.

The idealized steps the student takes up to this point are preparation for learning, but they do not accomplish it. These steps are the "practice" part of "learning through practice." The student would execute all, or most, of these steps even if she were only performing the skill (problem solving) and not attempting to learn at all. In the actual learning step, the student uses diagnostic information to attempt to *fix* parts of the knowledge that were used to generate an answer. If the student answers incorrectly, she usually focuses on trying to modify those *misconceptions* in factual or planning knowledge believed responsible for failure, so that on the next problem she will succeed. A variety of generalization or discrimination techniques may be used to effect the modifications (see e.g., Anderson, 1982; Dietterich & Michalski, 1981). A student who answers correctly may still use available diagnostic information to make her planning knowledge more efficient. Lines of reasoning that lead to dead ends can be examined for misconceptions, as can redundancies in the main solution line. Even an ideal solution line can yield improvements in knowledge, not by provoking misconceptions to be fixed so much as by helping *formative conceptions* to be solidified. For example, the student may compose sequences of rules (Anderson, 1982) into macro-operators, or may simply change the "strength" associated with rules or methods, so they are more likely to be chosen on future similar occasions.

As a concrete example of this abstract cycle, consider the student learning algebra by doing homework problems. The student's task is to solve the next problem in the book. His formative knowledge of algebra includes an understanding of algebra transformation rules (e.g., axioms such as $x + y = y + x$) and algebra planning methods, beliefs about how to manipulate expressions in order to achieve algebra problem solving goals (e.g., "If there is more than one instance of the unknown in the equation, consider using transformation rules that collect" (Bundy & Welham, 1980). For each question, the student must decide which planning methods and transformation rules are relevant to the current expression, and apply that knowledge to produce one or several steps of mathematical reasoning. This process repeats until the reasoning chain produces an answer.

Once the student has exercised her algebra skills in this fashion, she may learn from the process. If the student sees the answer is incorrect, can isolate the faulty

reasoning step, and can determine the specific misconception underneath the faulty step (e.g., a factual error might be the belief that $\sqrt{a + b} = \sqrt{a} + \sqrt{b}$), then she is in a position to modify the relevant algebra knowledge in a principled way. If the answer is correct, the student may still improve her algebra planning knowledge. The answer may contain extraneous steps that can be eliminated by more refined planning methods (e.g., One may refine the above selection heuristic, creating "If there is more than one instance of the unknown in the equation and the equation is of the form a < variable > + b < variable > . . . then consider using the Distributive rule for addition and multiplication"). A right answer without such wasted steps can also result in improvements to knowledge. For example, students often learn to compose rules in algebra. Well-practiced axioms, like the commutativity of addition, become composed with other axioms; you rarely see two successive steps in a proof that differ only through the application of $x + y = y + x$.

Why Learning Through Practice with Existing Tools is Often Ineffective

I think it is fair to say that people currently learn very slowly and inefficiently through practice. With even our modest characterization of learning we can begin to understand *why* it is so difficult. It is possible to isolate several specific points at which each of the major learning processes can go awry, and many reasons why a student may fail to learn from a particular practice task using existing educational tools. In turn, this understanding provides a basis for designing educational tools that better support those specific learning processes.

Consider the algebra student again. Both standard textbooks and programmed-learning systems may not elicit a high volume of student misconceptions or formative conceptions because they pose questions in a fixed sequence. It is haphazard, therefore, whether a given question will cause the student to *exercise* any mistaken or weak beliefs. Questions that only elicit well-practiced knowledge rarely lead to significant learning.

Even if a question elicits suspect conceptions, the student may fail to perform an adequate *diagnosis*, or *detect* the conception causing an incorrect response. For example, a question can elicit a mistaken belief that accidentally produces the correct answer; or it may cause multiple suspect misconceptions to be exercised, obscuring the actual mistaken one. Students may also fail to detect errors because traditional teaching methods delay information critical for diagnosis. Diagnostic processes work most effectively when critical information is available at the time of reasoning. If the student does a question one day, and is not told the answer is wrong until the next day (i.e., the teacher grades and returns the homework), he may have forgotten the reasoning that led to the mistake. Unless the student is now able and willing to painstakingly reconstruct his reasoning, he will have lost any chance to detect reasoning errors. Finally, questions that do expose misconceptions may do so inefficiently. One observes, for example, that

much of the time students spend answering many questions is actually spent making slips, losing track of where they are, regaining a line of thought, or practicing well-learned operations, not focusing on weak ones.

Once the student has invested the time to detect a mistaken belief, his effort may still be wasted because the sparse feedback provided by a textbook (usually they only give the correct answer) often fails to provide the required support to help the student correctly *fix* mistaken beliefs. Telling the student he is incorrect when he says $\sqrt{16 + 4} = 6$ may be enough to cause him to discard the belief $\sqrt{a + b} = \sqrt{a} + \sqrt{b}$. However, often the correct modification is not to throw out a belief, but to alter it slightly, or add a new transformation rule. For example, assume the student has the following mistaken belief about "cross multiplication": $\frac{e_1}{e_2} + \frac{e_3}{e_4} = e_5 = e_1e_4 + e_3e_2 = e_5$ (where e_i denotes any expression). If a question exposes this error to the student we would like a tutor to ensure that he does not simply eliminate the belief, but instead modifies it slightly to: $\frac{e_1}{e_2} + \frac{e_3}{e_4} = e_5 = e_1e_4 + e_3e_2 = e_5e_2e_4$. Simply telling the student his answer is correct or incorrect is rarely adequate support to successfully accomplish this change. Such sparse feedback is even less useful in the case where a student makes an error because he does not believe a certain valid transformation rule, rather than because he does believe an invalid rule. If a student does not know trigonometric identities such as $\sin(x) = y \equiv \text{ARCSIN}(y) = x$, saying he is wrong on questions requiring the use of these tautologies gives no information. The student will repeatedly make the same mistake.

Goals of Tools for Learning Through Practice

Our analysis indicates that learning through practice comprises several major processes that are difficult for students to complete successfully by themselves. Further, if any of the component processes fail, students are unlikely to learn anything from the task or question at hand. This analysis, coupled with insight into how existing educational aids fail to facilitate learning processes, suggests many goals for designing new tools. A few of them are:

- Maximizing the student's opportunity to exercise formative and flawed conceptions.
- Helping the student detect and diagnose the misconceptions that contribute to any sub-optimal performance, especially by providing immediate feedback information. The misconceptions include both flaws and omissions in planning knowledge, and flaws and omissions in knowledge of domain facts.
- Helping the student fix misconceptions, once they have been detected and characterized.

- Helping the student strengthen and make more efficient (e.g., by composing operations) correct but formative conceptions that have been exercised.

In the following sections I discuss examples of tools that fulfill some of these goals. The examples are by no means exhaustive; instead, they are meant to suggest the wide range of relatively simple solutions that are possible.

LEARNING TOOLS THAT HELP EXERCISE FORMATIVE AND FLAWED CONCEPTIONS

A first obvious step is to develop facilities to elicit and exercise more misconceptions and formative conceptions. Each question or task provides an opportunity to improve a facet of a complex cognitive skill only if we use that facet in arriving at an answer, and only if that facet has some weakness or flaw. One technique, therefore, would be to consistently provide tasks that are challenging for the student, ones that are likely to elicit flawed or formative conceptions. All good human tutors, whether in chess or algebra, try to pick problems at the student's level, for just this reason.

This sort of tailoring, however, is difficult for humans to do, and even more difficult to build into an automated tutoring environment. It requires the tutor to maintain a *student model*, a structure that represents the tutor's idea of formative knowledge of the student at a point in time. Although recent research in artificial intelligence has improved our ability to build student models (Burton & Brown, 1982; London & Clancey, 1982; Sleeman, 1982; Sleeman & Smith, 1981), there is currently no strong theory of how to rapidly and accurately induce such models from the student's overt performances. This research should be continued to the point where we can build tutors that are highly skilled at building student models. However, while pursuing this goal, we should not lose sight of the fact that it now is possible to design useful learning aids that are within the current state of the art.

Although it is difficult to elicit misconceptions by intelligently guiding students through questions that will exercise weaknesses in their formative knowledge, it is relatively easy to help them by making their own self-guided search more efficient. Specifically, a simpler approach to eliciting a higher volume of student misconceptions is to increase the rate at which students do questions or, more accurately, the speed with which they perform the reasoning steps that lead to an answer. If students can reason more conveniently, they can do more tasks in a given time, and should thus encounter more of their formative and erroneous conceptions.

One way to make reasoning more convenient and rapid is to provide the student with an assistant who can take care of the more mundane details of solving a problem, letting the student focus on the harder parts—the parts from

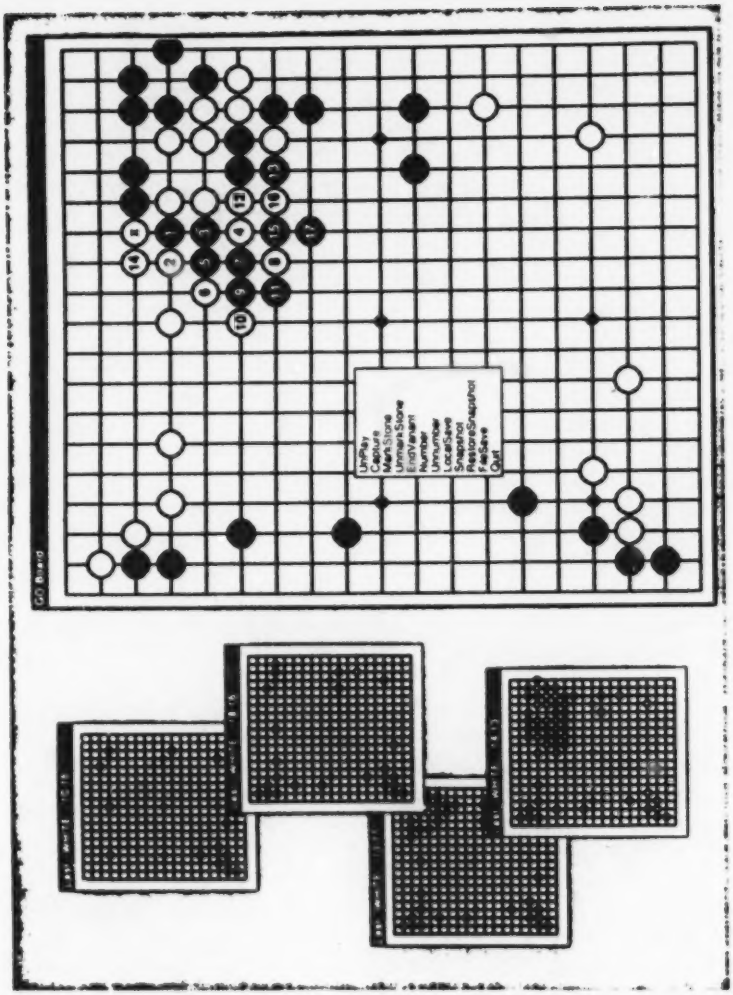

FIG. 13.1. The electronic Go board. To the right is the board on which games are played. Clicking the left mouse button on a location puts a black stone there, clicking the middle button puts a white stone at the location. The right mouse button brings up the menu shown in the center. Many of the options in the menu are explained in the text. To the left are several game icons. The positions they contain can be restored to the main board at any time.

BEST COPY AVAILABLE

which he or she could potentially learn. In algebra, again, a student solving for w in $2(3w + 55) + 2w = 870$ might go through the following reasoning steps: do the algebraic multiplication [$2(3w) + 2(55) + 2w = 870$]; do the numeric multiplication [$6w + 110 + 2w = 870$]; do the additions [$8w + 110 = 870$]; do the movement of terms [$8w = 760$]; and, finally, do the division [$w = 95$]. If the student is already an expert in numeric computations and is now practicing symbolic multiplication skills, only the first and second of these steps are germane to the *point* of the question. He or she can only exercise formative knowledge and acquire new knowledge on these steps. The rest of the steps exercise well-practiced skills; hence, much of the time spent answering the question does not contribute to learning. Worse still, if the student makes a slip in these well-learned steps, it may obscure a mistake in the first steps, thus taking away any chance at all of learning from the question. An automated assistant could alleviate these problems by doing the unimportant details of problem solving. A student interacting with a computerized algebra environment could construct the first steps of the solution, then select a mouse button, or menu item, to tell the assistant to "do the rest," or even "do until the next interesting step."

Although such an assistant would not have to maintain a complete model of the student, it would need to be an "algebra expert system." The assistant would need to know how to solve algebra problems from any point at which the student might want to stop. Fortunately, this technology, unlike student modeling, is within reach today. There are several programs (e.g., REDUCE and MACSYMA) that are very competent algebraists. It is relatively simple to interface them to a simple assistant who would pass them the equation left by the student and possibly some instructions about the form of the required result. GED is a graphical algebra editor that provides such a capability (McArthur, 1985). It is part of our ongoing project to develop an intelligent tutor for algebra, some aspects of which I discuss below.

Learning Tools with Limited Intelligence

Even a tool with no particular general or domain-specific expertise can provide significant learning aids. Consider an electronic Go board like the one shown in Fig. 13.1. Go is a complex game of strategy like chess. One player is black, the other white, and they alternate placing circular "stones" on the intersections of a 19 × 19 board. The electronic Go environment is not a Go playing program; it knows nothing about Go, or about students of Go. It only provides a way of allowing users to play stones (users select the intersection they want to play on by pointing and touching a mouse button) and a menu of options. The options, however, make the electronic Go board a much more powerful learning environment than a traditional Go board.

The LOCALSAVE and FILESAVE menu options allow the Go student to record and recall any game he or she wishes. The student therefore does not need

to waste time setting up game situations to learn from. More important, the MARKSTONE, NUMBER and ENDVARIANT options provide the student with a simple but powerful facility for exploring alternate lines of play. At any time, the student can number the stones in the order played, market one of the stones, then request that all stones played after the marked one be removed. He or she can therefore rapidly back up and try another alternative. The SNAPSHOT and RESTORESNAPSHOT options provide similar functionality. Using them, the student can take a "picture" of any interesting game situation, like the ones on the left side of Fig. 13.1, then recall these alternatives at any future time by simply pointing.

In the traditional gaming environment, this sort of learning by exploring alternatives is almost impossible. In the electronic Go environment, however, trying out many alternative hypotheses and exercising many pieces of formative Go knowledge is both low-cost (it takes almost no time) and high-safety (by trying new lines the student does not risk losing old, perhaps better, lines; they can be recalled at any time). The simple electronic Go board *encourages* the student to rapidly try out a wide range of skills, both well learned and formative.

The ability to explore a wide range of hypotheses is an important way to exercise misconceptions and learn in many areas. In most cases, creating an environment that encourages this activity is no more difficult than it was for Go. For example, a simple graphical interface, called GED (McArthur, 1985), provides this functionality for algebra. Figure 13.2 shows a student interacting with GED.

GED consists of several windows. The middle-left window is a workspace in which the student transforms algebraic expressions that represent problems he or she has been given. The lower-left window contains menus that effect the transformations. All transformations are done by pointing at and marking pieces of expressions in the workspace, then selecting an operation in the menu. To the right of the workspace is a commentspace window that displays textual feedback from the tutor to the student. We attempt to keep its use to a minimum. More important for our present purposes is the top window, or displayspace. This window contains a *reasoning tree*, recording all the reasoning steps the student has taken in attempting to solve a problem. The empty box at the bottom of one of the lines or branches represents the current *problem focus*. When the student completes the current reasoning step, by modifying the equation in the workspace, he or she will select NEW STEP on the lower-right options menu. This option will cause the current workspace expression to fill the empty box, and a new box will sprout below. The options menu contains a number of items that allow the student to effectively manipulate expressions and to communicate with the tutor. Not all the options are yet functional, but we discuss several of the working items below.

The reasoning tree is analogous to the linear list of equations the student traditionally writes in a notebook. However, together with the options menu, the reasoning tree provides a much more powerful tool for supporting algebraic

learning than the traditional pencil-and-paper medium. If the student decides the current solution line is not profitable, or just wants to investigate a new line of attack on the problem, he or she selects GO BACK on the options menu. GED will then ask which solution expression in the displayspace the student wants to now be the problem focus. The student responds by using a mouse to point to any expression in the reasoning tree and selects it by clicking one of the mouse buttons. This expression then becomes the current expression in the workspace, and in the

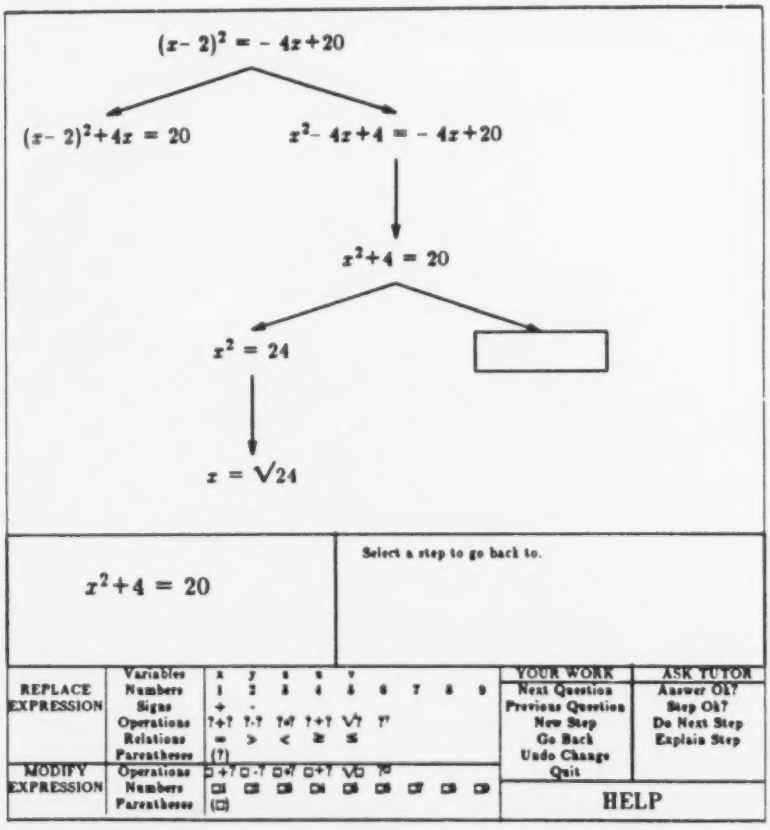

FIG. 13.2. The GED graphical display. The GED display comprises several windows. The top window, or displayspace, records the student's problem solving in the form of a reasoning tree. Below the displayspace is the workspace, where the student actually transforms the current expression to arrive at a new one, and a commentspace for textual feedback. The lower band of windows are menus. The student uses the selections from the left menu to edit or transform his current expression in the workplace. The options menu to the right allows the student and tutor to communicate about the problem.

displayspace it becomes the problem focus. A new empty box branches below the selected expression, creating a new solution line for the student to investigate. For example, in Fig. 13.2, having been told the answer ($x = \sqrt{24}$) is wrong, the student has just clicked on GO BACK, then selected the equation $x^2 + 4 = 20$. GED then rearranged the reasoning tree, adding a new branch, with an empty expression, at this point.

GED'S GO BACK option is just one of the ways it facilitates learning algebra. It provides a way for the student to try out problem-solving strategies efficiently. The traditional pencil-and-paper medium does not encourage this exploration and learning because it exacts a high cost to trying multiple solution lines. To try a new line, students must erase the old one, which they are reluctant to do because it is slow, and because they may forget the line, if they wish to return to it. Thus the traditional pencil-and-paper medium tacitly encourages the students to think of such changes as *mistakes to be avoided*. On the contrary, the ability to try out hypotheses rapidly, especially incorrect ones, is central to learning.

Although many aspects of GED, and our algebra tutor in which it is embedded, will contain sophisticated expertise, the previous facilities require only a bitmapped-display terminal and a generous amount of cheap memory. The tools obtain their leverage not through intelligent software, but through simple software intelligently conforming to the processes behind cognitive skills. The above tools try to simplify the processes of mathematical reasoning, in order to promote the use of formative and flawed algebraic knowledge. The process of mathematical reasoning has two important features for our purposes: It is a branching search through a space of possible mathematical transformations, and it is *nonlinear* in nature. Students will often want to change the focus of problem solving by pursuing one line of reasoning, suspending it, pursuing another, then returning to the first line. The GED editor obtains most of its strength as a learning tool by simply providing an external medium that reflects these properties. GED supplies a representation of reasoning steps that explicitly captures the tree-like results of the reasoning process and provides a mechanism that allows the student to externally change the focus of problem solving. GED is actually a medium in which you can *do* reasoning, unlike pencil-and-paper, which merely *records* reasoning.

LEARNING TOOLS THAT HELP DETECT AND FIX MISCONCEPTIONS

A student who makes a mistake is in a position to learn, but still has a long way to go before he or she can acquire knowledge from the current task. Providing supports that help students isolate and fix flawed and formative conceptions, or add an omitted conception to their knowledge, is just as important as providing

tools to help elicit the misconception in the first place. As I have noted, the diagnosis and fixing of conceptual errors are the most difficult processes in learning through practice. Fortunately, although traditional educational media often do not aid these processes, interactive computer tools show great promise in doing so.

Again, many such tools would require the development of a highly intelligent, complex, system software, not only able to model the student, but also embedding a great deal of pedagogical expertise. Such expertise is necessary to decide when to intervene to help the student. You don't always interrupt students with advice when they make a mistake, or *only* when they make a mistake. Further, if you do interrupt the student, you must decide whether to do it immediately, or defer the interruption for a while. Finally, pedagogical expertise is necessary to determine what to say, once an interruption has occurred. As with student-modeling software, promising research has begun to formalize and test good pedagogical strategies (Brown, Burton, & de Kleer, 1982; Burton & Brown, 1982; Clancey, 1979); however, the development of a reliable tutor with such expertise is a long-term, not short-term goal. But, again, a variety of useful tools can be developed in the short term. Broadly, these tools take the form of *debugging* aids. They largely obviate the need for a sophisticated theory of when to stop the student and what to say, by leaving such decisions up to the student. Students are simply made aware of tools and invoke them at their discretion when they make a mistake or need help.

Consider again the task of the algebra student in Fig. 13.2 who has just answered the question incorrectly ($x = \sqrt{24}$). GED has encouraged the student to exercise much formative knowledge of algebra, but now, to learn from the task, she has to wade through all this easily generated reasoning to find the one step that hides a misconception. GED's options menu provides several tools that will help simplify this search. First, the STEP OK? menu item allows the student to ask the tutor if any step in the reasoning tree constitutes an appropriate mathematical transformation in the current context. The student selects STEP OK?, then points to any node in the reasoning tree. The REDUCE algebra expert system, which interfaces with GED, then says whether the step from the previous node to the selected one is acceptable. In critiquing the student's step, the tutor makes a distinction between steps that are mathematically invalid, and steps that are inappropriate. For example, in Fig. 13.2, the step from $(x - 2)^2 = -4x + 20$ to $(x - 2)^2 + 4x = 20$ is valid, but is inappropriate. It is not the one the algebra expert would choose, because it does not move the student closer to a solution. Thus, the student not only succeeds in isolating a faulty reasoning step, but also obtains a characterization of the misconception underlying the error. Invalid steps imply errors in knowledge of algebra transformation rules; inappropriate steps imply errors in algebra planning methods.

The student can use the STEP OK? operation any time he chooses. This freedom allows him to follow several problem-solving modes. For example, the

student can proceed in "careful" mode, checking the appropriateness of each step as it is made, or at the other extreme, he can operate in "reckless" mode, creating reasoning steps until an answer is obtained that is obviously incorrect. The student then studies the reasoning tree, regarding it somewhat like an engineer regards a malfunctioning electronic circuit. The engineer may take voltage or current measurements at any point to isolate the fault; the algebra student uses STEP OK? to take "appropriateness" measurements, to isolate the misconception.

DO NEXT STEP is similar to STEP OK?. Although STEP OK? lets the student confirm the validity and appropriateness of a step, DO NEXT STEP lets the student ask the tutor to generate the next step that an expert algebra problem solver would take from the current problem focus. I have already discussed this option as a means of providing an assistant that could help the student exercise more formative knowledge of algebra by supplying the mundane steps of a solution, leaving only the hard ones for the student. However, it is useful in helping the student to detect misconceptions, as well as to exercise them. The most obvious way to help a student understand an error is to tell him about it, as STEP OK? does. This method is not the only one, nor necessarily the best one. For example, instead of telling the student about the error, one might show the *logical consequences* of continuing this line of reasoning, letting the student draw the appropriate conclusions, if the consequences appear absurd. By repeatedly applying DO NEXT STEP, the student can ask the tutor to provide such consequences at any point in the reasoning tree.

The immediate utility of STEP OK? and DO NEXT STEP are that they quickly allow the student to pinpoint a single reasoning error in a large tree of possibilities, and to characterize the misconception behind the faulty step. More generally, and perhaps more importantly, these activities teach the student that an important part of learning a cognitive skill is *learning to study your own reasoning processes*. A surprisingly few students understand that reasoning can be explicitly examined, let alone that it can be debugged or improved. For example, when students are asked why they do poorly on a mathematics test in grade school, common answers are "I'm dumb in math" (girls usually say this), or "I had a bad test," or "The test was unfair" (mainly coming from the boys). Very few identified specific knowledge that they might have lacked, or even understood that their correctable lack of knowledge might have been responsible for failure!

The ability to show students that their reasoning can be studied derives from GED's reasoning tree, which attempts to explicitly represent the process and structure of students' thinking. By externalizing students' reasoning process we impress upon them that it is a bona fide entity. Equally important, by providing students with ways of probing, querying, and commenting on the reasoning tree, we show them that the reasoning process is a manipulable, fixable entity. In addition, externalizing students' reasoning reduces memory load and makes it simpler for students to perform effective manipulations on their reasoning.

More Learning Tools with Limited Intelligence

Just as tools that do not rely on complex expertise can be developed to help students exercise misconceptions, similar simple tools can be developed to help students detect and fix those misconceptions. The electronic Go board, in Fig. 13.1, for example, provides the student with the ability to detect the logical consequences of moves without requiring a built-in Go "expert system." One of the best ways to learn Go, or any other complex game, is to replay experts' games, always trying to guess the next move. At each point where the student picks a wrong move (not the expert's) he or she should try to determine why the move was wrong and the expert's was right. This exercise provides the diagnostic information necessary for the student to fix the misconceptions responsible for deviations from the correct (expert's) line of play. Unless the expert is standing over the student's shoulder, the only reasonable way to understand the mistake is for the student to look at the consequences of a selection by playing out the line following from the move and to compare this result with the consequences following from the expert's move. Using a traditional Go board, this process is usually so tedious that most students rarely come to a good understanding of their mistakes. With the electronic Go board, on the other hand, it is so simple to play out lines and retract them that I personally find myself finally understanding 90% of experts' moves instead of less than 50%.

Similarly, we could easily extend GED to provide several sources of useful information about reasoning that do not rely on having access to algebra expertise. For example, we may implement a new menu option called SHOW ANSWER MATCHING, which will allow the student to access her solution tree for any past question, enabling her to visually compare the path previously taken, to the solution now being generated. A student solving a problem may transform an equation into a familiar form, but may forget how to deal with that form just now. For example, the current equation might be $(x + 8)(x + 2) = -10x$, and the student has done expansions before. If the student now selects the SHOW ANSWER MATCHING item, GED can search back as far as it needs in her past problem-solving history to find an expression of the form (<variable> + <constant>)(<variable> + <constant>) = <constant>. It will then display the reasoning tree that records how the student solved the matching problem. This example demonstrates that the student does not need to revert to an automated algebra expert to isolate a wrong step. She can do this isolation by comparing some past performance with the present buggy behavior.

The ability to rapidly recall previous performance and graphically compare it to present reasoning can have many diverse roles in learning. For example, the tutor should help strengthen formative conceptions as well as help fix misconceptions. One of the most important facets of such learning involves composing operations. It is well known that in many domains one of the main ways expert and novice problem solvers differ is that the former have *chunked* (Anderson, 1982; Chase & Simon, 1978) together operations that often follow one another in

solving a problem. This composition results in complex schemas that the expert can automatically invoke when shown a new problem. This more structured knowledge not only lets the expert solve familiar problems more rapidly, but also allows the expert to identify and devote more time to reasoning about the non-standard aspects of more difficult problems. A major goal for a student learning a cognitive skill is, therefore, to identify which of the contiguous operations used in solving a problem should actually be composed, or remembered as a unit. The trouble is that not all contiguous operations can be usefully composed. The student needs to identify which operations have been *repeatedly* used together in correct solutions. An ideal way to simulate such learning would thus be to give the student a way of looking for *patterns of operations* in past problem solving. GED's complete record of the student's performance history and its graphical presentation of reasoning are ideal for this purpose. Not only is the student given all the information necessary to derive useful composition patterns, but the graphical presentation dramatically simplifies the student's job of searching the information and comparing lines. This tool, along with many other potentially useful ones, should be very simple to implement. All will rely on the complete audit trail of student editing activities that GED now retains. Such a record is trivial to maintain using an electronic algebra problem-solving medium, but impossible with the passive pencil-and-paper medium.

MOTIVATION AND LEARNING

Our discussion has ignored one important variable in learning. Obviously, techniques that increase the *motivation* of students will increase the rate at which cognitive skills are learned. Motivation seems to have a generally positive effect on all the important processes of learning cognitive skills. A highly motivated student works through more questions, and more rapidly, so will expose more of weak ideas and misconceptions. He will also spend more time doing the hard work of detecting and fixing misconceptions, even if no learning aids are available.

Considerable effort has recently gone into adding fantasy or "bells-and-whistles" features to educational software, in hope of heightening motivation. I have not emphasized these features for several reasons. First, although they certainly make learning more fun, these features have not been proved to make learning much better. Second, I believe that the sort of *responsive* and *reactive* learning tools outlined here themselves heighten motivation by appealing to important cognitive determinants of motivation.

Malone (1980) cites several such determinants of motivation, including a variable level of difficulty, multiple levels of goals, and the ability to control the environment. I believe another important cognitive determinant of motivation is

comprehensibility. I feel a common classroom syndrome begins when otherwise average students miss key concepts in class and start to perform poorly. The students may not even be aware of why they are performing poorly, and rarely ask for help. Lack of success and failure to comprehend subsequent concepts cause the students to be less and less motivated to spend time solving problems, setting up a vicious circle. On the surface such students may appear simply not interested in mathematics. However, what they require is not merely to become more interested; they need an environment that will assist in identifying and overcoming their specific cognitive deficits and that will reduce their failure rates. In this respect, I believe the tools I have discussed can be highly motivating educational devices.

CONCLUSIONS

Computers have the potential to enhance the learning and doing of many cognitive skills, but have not yet lived up to that potential. Even the most successful computer software tools are not as useful as they could be. Text editors, for example, are just that; they provide aids for the processes of adding and moving text, but they do not provide significant tools to help the processes of writing papers or turning *thoughts into words*. (See Brown, 1984, for some ideas for tools along these lines.)

One line of thought, exemplified in artificial intelligence research, is that the way to make computers useful for learning is to develop computer tutors embedding vast amounts of human intelligence. Computer tutors are intended to capture much, if not all, of the functionality of human teachers, including the ability to inductively construct models of the student's reasoning processes, to guide the student through chosen questions and through the process of reasoning, and to carry out extensive natural-language dialogues with the student. Although this approach will ultimately lead to useful learning environments, I have argued that the development of simpler computer tools represents an equally effective approach. Unlike intelligent computer tutors, these computer tools do not actively guide the student through the large space of alternatives that must be considered. However, by conforming to the specific learning processes that students may have difficulty in completing successfully by themselves, these tools can make the student's self-guided search much more effective.

ACKNOWLEDGMENTS

The research reported here has been supported in part by the National Science Foundation (Applications of Advanced Technologies Program), Grant No. MDR-8470342.

REFERENCES

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89,* 369–406.

Brown, J. S. (1984). Process versus product: A perspective on tools for communal and information electronic learning. In *Education and the electronic age.* Proceedings of a conference sponsored by the Educational Broadcasting Company.

Brown, J. S., & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science, 2,* 155–192.

Brown, J. S., Burton, R. R., & de Kleer, J. (1982). Knowledge engineering and pedagogical techniques in SOPHIE I, II, and III. In D. H. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 227–282). London: Academic Press.

Brown, J. S., & Van Lehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science, 4,* 379–426.

Bundy, A., & Welham, B. (1980). Using meta-level inference for selective application of multiple rewrite rule sets in algebraic manipulation. *Artificial Intelligence, 16,* 189–211.

Burton, R. R., & Brown, J. S. (1982). An investigation of computer coaching. In D. H. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 79–98). London: Academic Press.

Chase, W. G., & Simon, H. A. (1978). Perception in chess. *Cognitive Psychology, 4,* 55–81.

Clancey, W. J. (1979). Tutoring rules for guiding a case method dialogue. *International Journal of Man Machine Studies, 11,* 25–49.

Dietterich, T. G., & Michalski, R. S. (1981). Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods. *Artificial Intelligence, 16,* 257–294.

London, B., & Clancey, W. J. (1982). Plan recognition strategies in student modeling: Prediction and description. In *Proceedings of the 1982 National Conference on Artificial Intelligence* (pp. 335–338).

Malone, T. W. (1980). *What makes things fun to learn? A study of intrinsically motivating computer games.* Xerox Cognitive And Instructional Series Report CIS-7 (SSL-80-11).

McArthur, D. (1985). GED: An easy-to-learn graphical editor for algebraic expressions. In preparation.

Shavelson, R. J. (1981). Teaching mathematics: Some contributions of cognitive research. *Educational Psychologist, 16,* 23–44.

Sleeman, D. H. (1982). Assessing aspects of competence in algebra. In D. H. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 185–200). London: Academic Press.

Sleeman, D. H., & Smith, M. J. (1981). Modeling student's problem solving. *Artificial Intelligence, 16,* 171–187.